#5

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日
Date of Application: ２００１年 ２月２８日

出 願 番 号
Application Number: 特願２００１－０５５９９６

出 願 人
Applicant(s): インターナショナル・ビジネス・マシーンズ・コーポレーション

CERTIFIED COPY OF PRIORITY DOCUMENT

２００１年 ６月２１日

特 許 庁 長 官
Commissioner,
Japan Patent Office

及川耕造

【書類名】　特許願

【整理番号】　JP9000420

【提出日】　平成13年 2月28日

【あて先】　特許庁長官　殿

【国際特許分類】　G06F　5/06

【発明者】

【住所又は居所】　神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

【氏名】　川人　基弘

【発明者】

【住所又は居所】　神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

【氏名】　小松　秀昭

【発明者】

【住所又は居所】　神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

【氏名】　ジョン・ファレイ

【特許出願人】

【識別番号】　390009531

【氏名又は名称】　インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】　100086243

【弁理士】

【氏名又は名称】　坂口　博

【代理人】

【識別番号】　100091568

【弁理士】

【氏名又は名称】　市位　嘉宏

【代理人】

【識別番号】　100106699

【弁理士】

【氏名又は名称】　渡部　弘道

【復代理人】

【識別番号】　100104880

【弁理士】

【氏名又は名称】　古部　次郎

【選任した復代理人】

【識別番号】　100100077

【弁理士】

【氏名又は名称】　大場　充

【手数料の表示】

【予納台帳番号】　081504

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9706050

【包括委任状番号】　9704733

【包括委任状番号】　0004480

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　プログラムの最適化方法及びこれを用いたコンパイラ

【特許請求の範囲】

【請求項１】　プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度を向上させることができるかを解析するステップと、

前記解析の結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成するステップと

を含むことを特徴とするプログラムの最適化方法。

【請求項２】　前記パスを生成するステップは、

前記プログラムを実行することにより、前記解析の結果における前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、

得られた統計情報に基づいて、前記パスを生成するステップとを含む請求項１に記載のプログラムの最適化方法。

【請求項３】　プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムを実行することにより、前記プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、

得られた統計情報に基づいて、コンパイル結果として所定の命令のパラメータを特定の状態に固定したパスを含む機械語プログラムを生成するステップと

を含むことを特徴とするプログラムの最適化方法。

【請求項４】　前記統計情報に基づいてパラメータを特定の状態に固定できない場合に、コンパイル結果としてパラメータの状態を固定したパスを含まない機械語プログラムを生成するステップをさらに含む請求項３に記載のプログラムの最適化方法。

【請求項５】　オブジェクト指向プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法にお

いて、

前記プログラムにおける命令のうちで、メソッドの呼び出し先を限定でき、かつ当該メソッドの呼び出し先を限定することにより処理が速くなる命令を検出するステップと、

検出された前記命令におけるメソッドの呼び出し先を、当該命令の処理が速くなるように限定したパスを生成するステップと

を含むことを特徴とするプログラムの最適化方法。

【請求項６】　プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムにおける命令のうちで、変数の値を所定の定数値に限定でき、かつ当該定数値に限定することにより処理が速くなる命令を検出するステップと、

検出された前記命令における変数を、前記定数値に固定したパスを生成するステップと

を含むことを特徴とするプログラムの最適化方法。

【請求項７】　プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うコンパイラにおいて、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部と、

前記影響度解析部による解析結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化処理部と

を備えることを特徴とするコンパイラ。

【請求項８】　前記プログラムを実行した場合に、前記影響度解析部にて解析された前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいて当該命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部をさらに備え、

前記特殊化処理部は、前記命令のパラメータを、前記特殊化データ選択部によ

り決定された状態に固定した特殊化されたパスを生成する請求項７に記載のコンパイラ。

【請求項９】　前記特殊化処理部は、前記プログラム中に、プログラムの実行状態に応じて、前記特殊化されたパスと特殊化されていないパスとを選択的に実行するための分岐処理を生成し、

前記特殊化データ選択部は、前記分岐処理の挿入に基づく遅延を考慮して前記命令のパラメータをどの状態に固定するかを決定する請求項８に記載のコンパイラ。

【請求項１０】　プログラムのソースコードを入力する入力装置と、

入力された前記プログラムをコンパイルして機械語コードに変換するコンパイラと、

機械語コードに変換された前記プログラムを実行する処理装置とを備えたコンピュータ装置において、

前記コンパイラは、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度を向上させることができるかを解析する手段と、

前記解析の結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成してコンパイルを行う手段とを備え、

前記パラメータの状態を固定したパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力することを特徴とするコンピュータ装置。

【請求項１１】　プログラムのソースコードを入力する入力装置と、

入力された前記プログラムをコンパイルして機械語コードに変換するコンパイラと、

機械語コードに変換された前記プログラムを実行する処理装置とを備えたコンピュータ装置において、

前記コンパイラは、

前記プログラムを実行した場合に、前記プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、当該統計の結果に基づいて

当該命令のパラメータをどの状態に固定するかを決定する手段と、

前記命令のパラメータを、決定された前記状態に固定した特殊化されたパスを生成してコンパイルを行う手段とを備え、

前記特殊化されたパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力することを特徴とするコンピュータ装置。

【請求項１２】　前記コンパイラは、

前記特殊化されたパスを生成することなくコンパイルを行う手段をさらに備え、

前記コンパイラにおける前記命令のパラメータの状態を決定する前記手段は、固定すべきパラメータの状態を決定できない場合に、前記特殊化されたパスを生成することなくコンパイルを行う前記手段により生成された、前記特殊化されたパスを含まないコンパイル結果である機械語コードに変換されたプログラムを出力する請求項１１に記載のコンピュータ装置。

【請求項１３】　コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、

前記実行プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該実行プログラムの実行速度を向上させることができるかを解析する機能と、

前記解析の結果に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と

を前記コンピュータに実現させることを特徴とする支援プログラム。

【請求項１４】　コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、

前記実行プログラムを実行することにより、前記実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、

得られた統計情報に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と

を前記コンピュータに実現させることを特徴とする支援プログラム。

【請求項１５】　コンピュータを制御して実行プログラムの生成を支援する

支援プログラムを当該コンピュータの入力手段が読取可能に記憶した記憶媒体において、

前記支援プログラムは、

実行プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該実行プログラムの実行速度を向上させることができるかを解析する機能と、

前記解析の結果に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と

を前記コンピュータに実現させることを特徴とする記憶媒体。

【請求項１６】　コンピュータを制御して実行プログラムの生成を支援する支援プログラムを当該コンピュータの入力手段が読取可能に記憶した記憶媒体において、

前記支援プログラムは、

前記実行プログラムを実行することにより、前記実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、

得られた統計情報に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と

を前記コンピュータに実現させることを特徴とする記憶媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、コンピュータプログラムの最適化方法に関し、特に実行時のデータの性質に基づくプログラムの特殊化を行う最適化方法に関する。

【０００２】

【従来の技術】

通常、プログラミング言語で記述されたプログラムのソースコードをコンパイルする際、コンピュータにおける実行速度の向上を図るために、当該プログラムの最適化を行っている。

最適化の手法には種々の方法があるが、オブジェクト指向プログラミング言語

で記述されたプログラムにおいては、プログラムの実行時のデータの性質に基づいてプログラムを特殊化（specialization）することによる最適化手法がある。

【０００３】

従来のプログラムの特殊化の手法としては、１．スタティック・コンパイラ上でコンパイル時のみ行われる特殊化と、２．スタティック・コンパイラ上でバインディング（Binding）時及び実行時に行われる特殊化とがある。

前者の例としては、

Flow-Sensitive Interprocedural Constant Propagation :
PLDI(Programming Language Design and Implementation)'95

があり、後者の例としては、

Efficient Incremental Run-Time Specialization for Free : PLDI'99

がある。

【０００４】

スタティック・コンパイラ上でコンパイル時のみ行われる特殊化では、まず、プログラム内のメソッドコールの引数について解析し、「必ず同じ定数値がくると判断できる引数」を検出する。そして、検出された「必ず同じ定数値がくると判断できる引数」を用いて特殊化を行う。

【０００５】

一方、スタティック・コンパイラ上でバインディング（Binding）時及び実行時に行われる特殊化では、ループネストに特化してプログラムを特殊化する。すなわち、各ループネストのボディ（body）内で不変とわかる変数について特殊化を行う。この方法は、ネストごとにプログラムを特殊化するという特徴はあるが、基本的には上記のコンパイル時のみ行う手法と同様に、メソッドコールを行う場所で特殊化したメソッドコールを行う。

【０００６】

【発明が解決しようとする課題】

しかしながら、上述した従来のプログラムの特殊化による最適化手法は、いずれもメソッドコールを行う場所で呼び出しメソッドを特殊化するため、引数のパターンに対応する多くの特殊化したメソッドが必要となり、この特殊化したメソ

ッドに対応する実行コードが生成される。このため、コンパイル時間が増加し、かつメモリの消費量が増大していた。

【０００７】

また、上記従来のプログラムの特殊化による最適化手法は、上述したコンパイル時間の増加、メモリ消費量の増大といった観点から、ＪａｖａにおけるＪＩＴコンパイラ（Just In Timeコンパイラ）のようなダイナミック・コンパイラ（プログラムの実行時に動作するコンパイラ）には適用することが困難であった。

【０００８】

さらに、Ｊａｖａで用意されているダイナミック・コール（Dynamic Call）のような機能を用いたプログラムでは、メソッドコールを行う場所で呼び出しメソッドが特定できない場合が多い。そのため、メソッドコールを行う場所で呼び出しメソッドを特殊化する上記従来のプログラムの特殊化は実用的ではなかった。

【０００９】

そこで、本発明は、特殊化による最適化を含むコンパイルにおいて、コンパイルに要する時間を短縮し、コンパイル時のメモリ消費量を抑えることを目的とする。

また、本発明は、メソッドコールを行う場所で呼び出しメソッドが特定できない場合であっても特殊化を実行可能とすることを他の目的とする。

【００１０】

【課題を解決するための手段】

上記の目的を達成するため、本発明は、プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、最適化の対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度を向上させることができるかを解析するステップと、この解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成して特殊化するステップとを含むことを特徴とする。

本発明では、この所定の命令のパラメータを特定の状態に固定したパス（特殊化されたパス）と、この所定の命令のパラメータを特定の状態に固定しないパス

（特殊化されていないパス）とを、分岐処理により選択的に実行させるように、プログラムの最適化を行う。これにより、特殊化されたパスを実行した際にプログラムの実行速度を向上させることができ、かつこの特殊化されたパスを実行する割合が大きいならば、プログラム全体の実行効率を向上させることができる。

【００１１】

ここで、このパスを生成するステップは、 このプログラムを実行することにより、この解析結果における前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、得られた統計情報に基づいて、この所定の命令のパラメータを特定の状態に固定したパスを生成するステップとを含む。

このように、統計処理を行うことにより、特殊化されたパスを実行する確率がどの程度あるかを的確に調べることができる。

【００１２】

また、本発明は、プログラムの最適化方法において、最適化の対象であるプログラムを実行することにより、このプログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、得られた統計情報に基づいて、コンパイル結果として所定の命令のパラメータを特定の状態に固定したパスを含む機械語プログラムを生成するステップとを含むことを特徴とする。

【００１３】

さらに、この最適化方法は、この統計情報に基づいてパラメータを特定の状態に固定できない場合に、コンパイル結果としてパラメータの状態を固定したパスを含まない機械語プログラムを生成するステップを含むことができる。

【００１４】

また、本発明は、上記のような特殊化によるプログラムの最適化方法において、最適化の対象であるプログラムにおける命令のうちで、メソッドの呼び出し先を限定でき、かつこのメソッドの呼び出し先を限定することにより処理が速くなる命令を検出するステップと、検出された命令におけるメソッドの呼び出し先を、この命令の処理が速くなるように限定したパスを生成するステップとを含むことを特徴とする。

【００１５】

あるいは、上記のような特殊化によるプログラムの最適化方法において、最適化の対象であるプログラムにおける命令のうちで、変数の値を所定の定数値に限定でき、かつこの定数値に限定することにより処理が速くなる命令を検出するステップと、検出された命令における変数を、この定数値に固定したパスを生成するステップとを含むことを特徴とする。

【００１６】

また、本発明は、プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うコンパイラにおいて、コンパイルの対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部と、この影響度解析部による解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化処理部とを備えることを特徴とする。

【００１７】

さらに、このコンパイラは、このプログラムを実行した場合に、この影響度解析部にて解析された命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいてこの命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部をさらに備える構成とすることができる。そして、特殊化処理部は、この命令のパラメータを、特殊化データ選択部により決定された状態に固定した特殊化されたパスを生成する。

【００１８】

ここで、この特殊化処理部は、このプログラム中に、プログラムの実行状態に応じて、特殊化されたパスと特殊化されていないパスとを選択的に実行するための分岐処理を生成し、特殊化データ選択部は、この分岐処理の挿入に基づく遅延を考慮してこの命令のパラメータをどの状態に固定するかを決定する構成とすることができる。

【００１９】

さらにまた、本発明は、次のように構成されたことを特徴とするコンピュータ装置として提供することができる。すなわち、プログラムのソースコードを入力

する入力装置と、入力されたこのプログラムをコンパイルして機械語コードに変換するコンパイラと、機械語コードに変換されたこのプログラムを実行する処理装置とを備える。また、このコンパイラは、このプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度を向上させることができるかを解析する手段と、この解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成してコンパイルを行う手段とを備える。そして、このコンパイラは、このパラメータの状態を固定したパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力する。

【００２０】

また、本発明は、次のように構成されたことを特徴とするコンピュータ装置として提供することができる。すなわち、プログラムのソースコードを入力する入力装置と、入力されたこのプログラムをコンパイルして機械語コードに変換するコンパイラと、機械語コードに変換されたこのプログラムを実行する処理装置とを備える。また、このコンパイラは、このプログラムを実行した場合に、このプログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、この統計の結果に基づいてこの命令のパラメータをどの状態に固定するかを決定する手段と、この命令のパラメータを、決定された状態に固定した特殊化されたパスを生成してコンパイルを行う手段とを備える。そして、このコンパイラは、この特殊化されたパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力する。

【００２１】

さらに、このコンパイラは、この特殊化されたパスを生成することなくコンパイルを行う手段をさらに備える構成とすることができる。そして、このコンパイラにおける命令のパラメータの状態を決定する手段は、固定すべきパラメータの状態を決定できない場合に、この特殊化されたパスを生成することなくコンパイルを行う手段により生成された、この特殊化されたパスを含まないコンパイル結果である機械語コードに変換されたプログラムを出力する。

【００２２】

また、本発明は、次のように構成されたことを特徴とするコンピュータプログラムとして提供することができる。すなわち、コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、この実行プログラムにおける所定の命令のパラメータを特定の状態に固定することによりこの実行プログラムの実行速度を向上させることができるかを解析する機能と、この解析結果に基づいて、この実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とをコンピュータに実現させることを特徴とする。

【００２３】

あるいは、コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、この実行プログラムを実行することにより、この実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、得られた統計情報に基づいて、この実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とをコンピュータに実現させることを特徴とする。

【００２４】

さらに本発明は、かかるコンピュータプログラムをコンピュータの入力手段が読取可能に記憶した記憶媒体として提供したり、かかるコンピュータプログラムをネットワークを介して伝送させるプログラム伝送装置として提供したりすることもできる。

【００２５】

【発明の実施の形態】

以下、添付図面に示す実施の形態に基づいて、この発明を詳細に説明する。

図１は、本実施の形態におけるコンパイラの全体構成を説明する図である。

図１を参照すると、本実施の形態のコンパイラ１０は、入力プログラムに対して通常のコンパイル処理を行う汎用コンパイル処理部１１と、汎用コンパイル処理部１１によるコンパイル結果に対して影響度解析（Impact Analysis）を行う影響度解析部１２と、影響度解析部１２による解析結果を用いて特殊化を行うデータを選択する特殊化データ選択部１３と、特殊化データ選択部１３により選択されたデータを特殊化してオブジェクトプログラムを生成する特殊化コンパイル

処理部１４とを備える。

図１に示したコンパイラ１０の各構成要素は、コンピュータプログラムにより制御されたＣＰＵにて実現される仮想的なソフトウェアブロックである。ＣＰＵを制御する当該コンピュータプログラムは、ＣＤ－ＲＯＭやフロッピーディスクなどの記憶媒体に格納して配布したり、ネットワークを介して伝送したりすることにより提供される。

【００２６】

上記の構成要素において、汎用コンパイル処理部１１は、入力プログラムであるソースプログラムを入力し、通常のコンパイル処理を行って機械語コードによるオブジェクトプログラムを生成する。コンパイラ１０をＪａｖａにおけるＪＩＴコンパイラにおいて実現する場合は、ソースプログラムはバイトコードで記述されたプログラムとなる。

【００２７】

影響度解析部１２は、汎用コンパイル処理部１１によるコンパイル結果であるオブジェクトプログラムを対象として、影響度解析処理を実行し、プログラム内のデータにおける影響度情報を作成する。影響度解析処理とは、「どのプログラム内のデータを、どのように特殊化すると、より強力な最適化を行えるか」を調べる処理である。影響度情報には、この解析結果が記述される。影響度解析処理の具体的な内容については後述する。

【００２８】

特殊化データ選択部１３は、影響度解析部１２による影響度解析処理の結果、特殊化による影響のあるデータが存在する場合に、特殊化を行うデータを選択する。具体的には、汎用コンパイル処理部１１で生成されたオブジェクトプログラムを複数回実行し、影響度解析部１２で作成された影響度情報についてデータの出現頻度の統計を取る。すなわち、影響度解析処理の結果において特殊化による影響があると判断されたデータが、どのように実行される確率が、どの程度有るかを調べる。そして、得られた統計情報において特殊化を行うことにより一定以上の効果が得られると評価できるデータを特殊化の対象に決定する。特殊化の対象となるデータを選択する処理の具体的な内容については後述する。

ここで、統計を取るためには、当該プログラムが実際に実行されることが必要である。すなわち、当該プログラムを実行するシステムは、初期的には、汎用コンパイル処理部１１による通常のコンパイルを経たオブジェクトプログラムを実行する。そして、十分な回数実行して統計が得られた後に（統計を取るための実行回数は任意に設定できる）、特殊化の対象となるデータの選択が行われることとなる。

ただし、この統計を取るための実行は、当該統計を取ることを目的とした試験的な実行であっても良い。この場合、当該プログラムが実際のシステムで使用される場合には最初から適切な特殊化が行われた状態でコンパイルされることとなる。

【００２９】

なお、影響度解析部１２による影響度解析処理の結果、特殊化による影響のあるデータが存在しない場合、特殊化データ選択部１３は、上述した統計処理を行うことなく、汎用コンパイル処理部１１により生成されたオブジェクトプログラムを出力する。

また、特殊化データ選択部１３による統計処理の結果において、特殊化すべきデータが存在しない場合、またはプログラムの実行回数が統計処理のために設定された回数に満たない場合は、特殊化コンパイル処理部１４に移行して特殊化を行うことなく、汎用コンパイル処理部１１により生成されたオブジェクトプログラムを出力する。

【００３０】

特殊化コンパイル処理部１４は、特殊化データ選択部１３により特殊化の対象として選択されたデータを特殊化し、特殊化されたオブジェクトプログラムを生成して出力する。特殊化は、プログラム中の所定の命令におけるパラメータを特定の状態（定数化や呼び出し先の特定など）に固定することにより、パラメータの状態を判断する処理のようなパラメータが複数の状態を取ることに伴う処理を省略するものであり、対象であるデータの種類や性質に応じて個別的に実行される。例えば、命令中の定数を最も頻出する値に固定したり、メソッドの呼び出し先を最も多い呼び出し先に特定したりするといった処理が行われる。

【００３１】

図２は、本実施の形態によるコンパイラ１０の全体的な処理の流れを説明するフローチャートである。

図２を参照すると、まず、ソースプログラムが汎用コンパイル処理部１１に入力され、当該ソースプログラムに対して通常のコンパイル処理が実行される（ステップ２０１）。

次に、汎用コンパイル処理部１１によるコンパイル結果であるオブジェクトプログラムに対して、影響度解析部１２により影響度解析が行われる（ステップ２０２）。そして、特殊化による影響のあるデータが存在しない場合は、汎用コンパイル処理部１１により生成されたオブジェクトプログラムが出力されて、処理が終了する（ステップ２０３）。この場合、当該プログラムを実行するシステムは、汎用コンパイル処理部１１のみによってコンパイルされたオブジェクトプログラムを実行することとなる。

【００３２】

影響度解析において特殊化による影響のあるデータが存在すると判断された場合は、次に、特殊化データ選択部１３により、影響度解析処理の結果において特殊化による影響があると判断されたデータに対する統計処理が行われ、特殊化の対象となるデータの選択が行われる（ステップ２０３、２０４）。そして、特殊化すべきデータが存在しない場合は、汎用コンパイル処理部１１により生成されたオブジェクトプログラムが出力されて、処理が終了する（ステップ２０５）。

ここで、統計処理の結果、特殊化すべきデータが存在しない場合は、当該プログラムを実行するシステムは、汎用コンパイル処理部１１のみによってコンパイルされたオブジェクトプログラムを実行することとなる。

一方、プログラムの実行回数が統計処理のために設定された回数に満たないことにより特殊化すべきデータが選択されない場合は、当該プログラムがさらに実行されて統計処理を行うに十分な回数となるのを待つ。

【００３３】

プログラムの実行回数が統計処理のために設定された回数に達し、統計が取られた結果、特殊化すべきデータが存在する場合は、次に、特殊化コンパイル処理

部１４により、当該データの特殊化が行われる（ステップ２０５、２０６）。

これにより、当該プログラムを実行するシステムは、特殊化コンパイル処理部１４により特殊化されたオブジェクトプログラムを実行することとなる。

【００３４】

次に、上述した影響度解析部１２による影響度解析処理の内容及び特殊化データ選択部１３による特殊化の対象となるデータを選択する処理の内容について説明する。

ここでは、Ｊａｖａ言語に対して本実施の形態を適用した場合を例として、パラメータを特殊化する例と、グローバルデータの両方を特殊化する例の２つを挙げる。なお、以下の説明において、ALLTYPEとは実装を行っている全ての特殊化するタイプを意味する。図３にこの特殊化するタイプの例を示す。

【００３５】

図３に示す例では、特殊化のタイプとして、「定数」、「Ｎｕｌｌではない」、「クラスの特定」、「配列以外のクラスである」といったタイプが明示されている。ここで、「定数」は、プログラムにおける所定の命令中の定数を一定の値に固定する特殊化を行う。「Ｎｕｌｌではない」は、変数の値がＮｕｌｌとなる可能性があり、かつＮｕｌｌであれば処理が速くなる場合に、当該変数の値をＮｕｌｌに固定する特殊化を行う。「クラスの特定」は、メソッドの呼び出し先を限定でき、かつメソッドの呼び出し先を限定することにより処理が速くなる場合に、当該メソッドの呼び出し先を特定する特殊化を行う。「配列以外のクラスである」は、プログラムがＪａｖａのようにメソッドの呼び出し先として配列クラスを取ることができるプログラム言語で記述されている場合に、当該メソッドの呼び出し先を配列クラスに特定する特殊化を行う。なお、特殊化のタイプは、プログラムの種類や実装するシステム、使用環境などに応じて種々のタイプを設定することができる。

【００３６】

また、図３の命令欄には、これらの特殊化を行う可能性のある命令の種類が記載されている。例えば、プログラム中に四則演算やオブジェクト以外の変数に対する比較命令が含まれていたならば、これらの命令は「定数」タイプの特殊化を

行うことにより処理効率が向上する可能性がある。同様に、プログラム中にINSTANCEOF、CHECKCAST、仮想的なメソッド呼び出し命令などが含まれていたならば、これらの命令は「クラスの特定」タイプの特殊化を行うことにより処理効率が向上する可能性がある。したがって、これらの命令は影響度解析部１２による影響度解析処理の対象となる。

【００３７】

（１）パラメータの特殊化

まず、パラメータを特殊化する場合における本実施の形態の動作アルゴリズムを説明する。

最初に、汎用コンパイル処理部１１が、入力されたソースプログラムに対して通常の（特殊化を行わない）コンパイル処理を行う（図２、ステップ２０１参照）。このとき、ＵＤ－ｃｈａｉｎ（use-definition chain：使用－定義連鎖）及びＤＵ－ｃｈａｉｎ（definition-use chain：定義－使用連鎖）も作成しておく。

【００３８】

次に、影響度解析部１２が、汎用コンパイル処理部１１によるコンパイル結果（特殊化されていないオブジェクトプログラム）に対して影響度解析処理を行う（図２、ステップ２０２参照）。

影響度解析処理としては、まず、プログラム中のパラメータ（変数）に対して、予め影響度解析の対象として設定された特殊化の各タイプ（図３参照）に対して、特殊化を行った場合における効果を見積もる。

【００３９】

ここで、特殊化を行った場合の効果について説明する。

特殊化を行うとは、ここでは、複数の値を取る可能性のあるパラメータを含む命令に関して、当該パラメータが特定の値を取るものとしてプログラムを書き換えることを意味する。これに伴い、当該パラメータが当該特定の値以外の値を取る場合に対応するため、当該命令の前にガードコードを挿入し、特殊化されていないパスへの分岐処理を行う必要が生じる。したがって、パラメータを特殊化しても特殊化したパスの実行速度が特殊化前と変わらない場合、ガードコードを挿

入した分だけプログラム全体の実行速度は遅くなる。

一方、特殊化したパスの実行速度が特殊化前よりも速くなるならば、ガードコードを挿入したことによる遅延分を越えてプログラム全体の実行速度が向上する可能性がある。

【００４０】

そこで、特殊化を行った場合の効果を見積もり、これとガードコードの挿入による遅延を基準とするしきい値Ａとを比較することにより、プログラム全体の実行速度に対する影響の度合いを調べる。この影響の度合いは、例えば、ガードコードの挿入による遅延に対する特殊化の効果の比で表すことができる（以下、この値を影響度値と呼ぶ）。この場合、影響度値が１であるとは、特殊化されたパスの実行速度の向上分がガードコードの挿入による遅延分と等しい（すなわち、プログラム全体の実行速度は変わらない）ことを意味する。

そして、特殊化されたパスの実行速度の向上がプログラム全体の実行速度に対して一定以上の影響を与える場合に、当該特殊化におけるパラメータ及び特定の値を、特殊化による影響のあるデータ（以下、特殊化対象データ）として影響度情報に記述する。また、影響度情報には、かかる特殊化におけるプログラム全体の実行速度に対する影響度値を記述する。

【００４１】

なお、しきい値Ａを適当に設定することにより、プログラム全体の実行速度に対してどの程度の影響を与える場合に特殊化対象データを影響度情報に記述するかを調節することができる。しきい値Ａを小さく設定すれば、わずかでもプログラム全体の実行速度を向上させる特殊化対象データが、影響度情報に記述されることとなる。一方、しきい値Ａを大きく設定すれば、相当程度プログラム全体の実行速度を向上させる特殊化対象データでなければ、影響度情報に記述されない。しかしこの場合、影響度情報に記述される特殊化対象データの数が減るので、特殊化データ選択部１３による統計処理のコストを削減し、特殊化データ選択部１３を実現するＣＰＵの負担を軽減することができる。

【００４２】

図４は、上述したパラメータに対する特殊化における影響度解析処理のアルゴ

リズムを説明する図である。

図４のアルゴリズムにおける６行目の

“Estimate(impact, p, p, φ);”

において特殊化の効果を見積もる。かかる処理の具体的な内容（アルゴリズム）は図５に示す。

【００４３】

また、図４のアルゴリズムにおける８、９行目の

“if (Impact[p][type] > しきい値Ａ)”
“Effective ∪= { p, type } ;”

において特殊化の効果としきい値Ａとを比較し、影響度情報に記述する情報を決定している。８行目のコメントにあるように、しきい値Ａの値を１とすれば、ガードコードの挿入による遅延分だけ特殊化されたパスの実行速度が向上する。したがって、プログラム全体の実行速度を向上させるには、しきい値Ａの値を１よりも大きくする必要がある。

【００４４】

次に、特殊化データ選択部１３が、影響度解析部１２による影響度解析処理の結果に基づいて、特殊化を実行するデータを選択する（図２、ステップ２０３、２０４参照）。

図６は、特殊化データ選択部１３による統計処理のアルゴリズムを説明する図である。

図６において、３行目にあるように、影響度情報に記述されたパラメータ（ｐ）に対して、対応する特殊化の方法（ｔｙｐｅ）に固有の条件で統計を取る。また、１行目にあるように、影響度情報（Ｅｆｆｅｃｔｉｖｅ）にデータが記述されていない場合（φ空集合）は、統計処理を行わない。

【００４５】

次に、特殊化データ選択部１３は、統計処理により得られた統計情報に基づいて、特殊化を実行するデータ（以下、特殊化実行データ）を選択する。

上述したように、特殊化とは、複数の値を取る可能性のあるパラメータを含む命令に関して、当該パラメータが特定の値を取るものとしてプログラムを書き換えることである。したがって、パラメータが取り得る値は複数あり、その個数だけ特殊化のバリエーションが存在することとなる。そこで、特殊化のバリエーションごとに出現頻度の統計を取り、パラメータが取り得る値のうちで最も頻出する値を選び、これを特殊化実行データとする。

ただし、最も頻出する値であっても、全体に対して十分大きな確率で出現しなければ、プログラム全体の処理効率が向上しないと考えられる場合がある。すなわち、パラメータの取り得る値が多数ある場合、各値が同程度の確率で出現するならば、最も頻出する値であっても出現率が大きくならない場合があり得る。この場合、特殊化を行ったとしても、特殊化されていないパスが実行される確率が高いため、ガードコードを挿入したことによる遅延がプログラム全体の実行速度に大きく影響することとなる。

したがって、ここでは、パラメータが取り得る値のうちで最も頻出する値について、このパラメータがこの最頻出値を取る確率を統計情報から取得し、この最頻出値が出現する確率と、この最頻出値で特殊化した場合における影響度値とを乗じた値を算出する。そして、得られた乗算値とガードコードの挿入による遅延を基準とするしきい値Ｂとを比較することにより、プログラム全体の処理効率が向上するかどうかを調べる。これにより、一定以上の効果が得られる場合に、当該最頻出値を上述した特殊化実行データとして最終的に選択する。

【００４６】

なお、しきい値Ｂを適当に設定することにより、特殊化によるプログラムの最適化が当該プログラム全体の実行効率に対してどの程度の影響を与えるかを制御することができる。しきい値Ｂを小さく設定すれば、上述した考え方に基づいて、特殊化を行わない場合よりもわずかでも実行効率が上がると考えられる場合に特殊化を行うこととなる。一方、しきい値Ｂを大きく設定すれば、相当程度プログラムの実行効率を向上させると考えられる場合でなければ特殊化を行わない。

しかしこの場合、特殊化を行う箇所を特殊化の効果の大きい場所に制限することとなるため、コンパイルの際のＣＰＵの負担を軽減し、かつ効果の大きい最適化を実現することができる。そのため、しきい値Ｂを適切に設定することにより、コンパイルに要する時間をあまり増大させることなく特殊化を実行することができ、ＪａｖａにおけるＪＩＴコンパイラのような、プログラムを実行する際にコンパイルを行うダイナミック・コンパイラに対しても適用することが可能となる。

【００４７】

特殊化データ選択部１３により上記のようにして決定される特殊化実行データと、影響度解析部１２により決定される特殊化対象データとを比較すると、影響度解析部１２により決定される特殊化対象データは、特殊化することによってプログラム全体の実行速度の向上が見込まれる全てのパラメータに関するデータであり、特殊化データ選択部１３により決定される特殊化実行データは、特殊化対象データのうちでプログラム全体の実行効率を最も向上させるパラメータに関するデータである。

【００４８】

図７は、上述したパラメータに対する特殊化における特殊化実行データを選択する処理のアルゴリズムを説明する図である。

図７のアルゴリズムにおける３行目の

“odds = { p, type } に関する統計の中で，もっとも優秀だった確率;”

において、パラメータが取り得る値の最頻出値の出現率を求める。また、４行目の

“val = oddsに対応する { p, type } の値;”

において、最頻出値自体を求める。そして、５、６行目の

“if (Impact[p][type] * odds > しきい値Ｂ)”

“Specialize ∪= { p, type, val } ;”

において、最頻出値の出現率とこの最頻出値に対応する影響度値とを乗算し、その乗算値としきい値Ｂとを比較し、特殊化を実行するデータを決定している。

なお、９行目にあるように、所定のパラメータに関して特殊化を実行するデータ（Ｓｐｅｃｉａｌｉｚｅ）が決定されなかった場合（φ空集合）は、当該パラメータに対する特殊化コンパイル処理部１４による特殊化は行わない（図２、ステップ２０５参照）。

【００４９】

最後に、特殊化コンパイル処理部１４が、特殊化データ選択部１３により選択された特殊化実行データに対して、実際に特殊化を実行する。特殊化は、上述したように、パラメータの種類や特殊化実行データの値に応じて個別的に実行される。

【００５０】

以上のように、本実施の形態では、特殊化によりプログラム全体の実行速度が向上する程度とかかる特殊化されたプログラムが実行される頻度とに基づいて、当該プログラム全体の実行効率を考える。このため、従来の特殊化において、所定のパラメータが必ず定数値を取る場合など極めて限定された場合でのみ特殊化が可能であったのに対し、本実施の形態は、特殊化によってプログラム全体の実行効率が一定以上向上すると考えられる場合には特殊化を行うことができる。

また、従来、プログラム中の特殊化できる可能性のある全てのパラメータに対して特殊化が可能かどうかを調べて特殊化を実行していたのに対し、本実施の形態は、特殊化コンパイル処理部１４により特殊化を実行する前に、影響度解析部１２による影響度解析処理を行い、さらに特殊化データ選択部１３により特殊化の対象となる値の実際の出現頻度を統計処理するという二重の処理により、特殊化の対象とするパラメータを絞り込んでいる。これによって、特殊化を行う場合のコンパイル時間及びメモリ消費量の増加を少なく抑えることができる。

さらに、本実施の形態は、メソッドコールを行う場所で呼び出しメソッドを特

殊化するのではなく、実際に呼ばれたメソッドについて解析して特殊化を行う。このため、メソッドコールを行う場所で呼び出しメソッドを特定できない場合であっても、特殊化を行うことができる。

【００５１】

次に、上述したパラメータの特殊化の適用例を説明する。

図８は、本適用例に用いるサンプルプログラム（以下、第１のサンプルプログラムと記す）を示す図である。

コンパイラ１０に図８の第１のサンプルプログラムが入力されると、まず、汎用コンパイル処理部１１により、当該第１のサンプルプログラムに対して通常のコンパイルが行われる。ただし、第１のサンプルプログラムは、この通常のコンパイルにおける最適化では、特に変更される箇所はない。なお、これ以降、第１のサンプルプログラムはコンパイルの過程で最適化が行われるため、コンパイル結果は機械語で記述されたオブジェクトプログラムとなっている。しかしながら、説明の簡単のため、ここではソースプログラムの状態のままで記述する。以下の他の適用例についても同様である。

【００５２】

次に、影響度解析部１２により、汎用コンパイル処理部１１によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。

図９は、図８に示した第１のサンプルプログラムに対する影響度解析処理の結果を示す図である。

図９（Ａ）を参照すると、パラメータ（図示のｐ欄）srcBeginが何らかの定数値を取る場合（「定数」タイプの特殊化を行った場合）に、影響度値２．２５が得られる。また、パラメータsrcEndが何らかの定数値を取る場合に、影響度値１．２５が得られる。したがって、図９（Ａ）に提示された第１のサンプルプログラム中の４つのパラメータのうち、srcBegin及びsrcEndの２つについて、定数値の出現頻度の統計を取ればよいことがわかる。そして、この２つのパラメータに関する情報が特殊化対象データとして影響度情報に記述される。

なお、図９（Ｂ）には、第１のサンプルプログラムのソースプログラム中にお

けるパラメータsrcBegin、srcEndの位置を示してある（太字で記し、下線を付してある）。

【００５３】

次に、特殊化データ選択部１３により、パラメータsrcBegin、srcEndに関して、取り得る値ごとに出現頻度の統計が取られる。

図１０は、得られた統計情報の例を示す図である。なお、図示の例は、第１のサンプルプログラムに対してSPECjvm98の中のJackベンチマークを使った場合における統計結果である。

【００５４】

次に、上記の統計情報に基づいて、特殊化データ選択部１３により、特殊化実行データが決定される。

図１０の統計結果を参照すると、srcBeginについては、定数値として値０を取る確率が１００％である。また、srcEndについては、定数値として値１を取る確率が６８％、値０を取る確率が１５％、値２を取る確率が４％、値３を取る確率が３％、値４を取る確率が３％である。その他の値については出現率が小さいので省略してある。

【００５５】

以上の統計情報によれば、srcBeginについては、特定の定数値の出現率が１００％であり、その影響度値が２．２５であるため、その乗算値は２．２５となる。この値はガードコードの挿入による遅延分の１よりも大きい。したがって、srcBeginは特殊化実行データとされる。

一方、srcEndについては、最も頻出する値１の場合で出現率が６８％であり、影響度値が１．２５であるため、その乗算値は０．８５となる。この値はガードコードの挿入による遅延分の１よりも小さい。したがって、プログラム全体の実行効率に対する影響は、srcBeginを特殊化することによる実行速度の向上よりもガードコードを挿入したことによる遅延の方が大きいこととなる。また、１以外の値では、出現率が小さくなるため、プログラム全体の実行効率に対する影響がさらに小さくなることは自明である。

したがって、この適用例では、srcBeginのみが特殊化実行データとなる。

【００５６】

以上の処理の後、特殊化コンパイル処理部１４により、図８に示した第１のサンプルプログラムにおけるsrcBeginが値０に特殊化されてコンパイルされる。

図１１は、図８の第１のサンプルプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【００５７】

次に、パラメータの特殊化の他の適用例を説明する。

図１２は、本適用例に用いるサンプルプログラム（以下、第２のサンプルプログラムと記す）を示す図である。

コンパイラ１０に図１２の第２のサンプルプログラムが入力されると、まず、汎用コンパイル処理部１１により、当該第２のサンプルプログラムに対して通常のコンパイルが行われる。

図１３は、図１２の第２のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。ここでは、ｃｈｅｃｋｃａｓｔの除去などの若干の最適化が実現されている。

【００５８】

次に、影響度解析部１２により、汎用コンパイル処理部１１によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。

図１４は、図１２に示した第２のサンプルプログラムに対する影響度解析処理の結果を示す図である。

図１４を参照すると、パラメータｅのクラスが特定されている場合に、影響度値３．９４が得られる。したがって、このパラメータｅについて、クラスの出現頻度の統計を取ればよいことがわかる。そして、このパラメータｅに関する情報が特殊化対象データとして影響度情報に記述される。

なお、図１３のプログラムにおいて太字で記し下線を付した部分が、パラメータｅを含み、特殊化による効果が見込まれる部分である。

【００５９】

次に、特殊化データ選択部１３により、パラメータｅに関して、取り得るクラ

スごとに出現頻度の統計が取られる。そして、得られた統計情報に基づいて特殊化実行データが決定される。具体的な統計情報の提示は省略するが、ここでは、パラメータｅがKeyEventというクラスである場合が最も出現率が高く、この場合の出現率と影響度値との乗算値はガードコードの挿入による遅延分の１よりも大きいものとする。したがって、パラメータｅは特殊化実行データとされる。

【００６０】

以上の処理の後、特殊化コンパイル処理部１４により、図１２に示した第２のサンプルプログラムにおけるパラメータｅのクラスがKeyEventに特殊化されてコンパイルされる。

図１５は、図１２の第２のサンプルプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【００６１】

さらにここで、特殊化のタイプとして「クラスの特定　かつ　Ｎｕｌｌでない」という組合せの状態が定義されている場合を考える。この場合、上述した各処理を経て、図１５における３行目の

“if (e != null)”

が特殊化されることとなる。図１６は、かかる特殊化が行われた状態のソースプログラムを示す図である。

【００６２】

（２）グローバルデータの特殊化

次に、グローバルデータを特殊化する場合における本実施の形態の同アルゴリズムを説明する。なお、実際のコンパイル処理においては、グローバルデータに対する特殊化と上述したパラメータに対する特殊化とを区別することなく実行するが、ここでは、説明上グローバルデータに対する特殊化のみを説明し、後述する適用例においてパラメータ及びグローバルデータに対する特殊化の例を示す。

【００６３】

最初に、汎用コンパイル処理部１１が、入力されたソースプログラムに対して

通常の（特殊化を行わない）コンパイル処理を行う（図２、ステップ２０１参照）。このとき、グローバルデータをメソッドローカルな変数に置き換えるScalar replacementの最適化を行うと共に、ＵＤ－ｃｈａｉｎ（使用－定義連鎖）及びＤＵ－ｃｈａｉｎ（定義－使用連鎖）の作成を行っておく。

【００６４】

Scalar replacementの最適化を行うアルゴリズムとして、

Lazy Code Motion : PLDI'92

に記載されたアルゴリズムがある。このアルゴリズムには、Down-Safetyという集合情報を計算する部分がある。これは、メソッド内の式を実行とは逆方向に動かす時に、その式の結果が変わらないと判断できる領域を集合として求めるものである。

したがって、Scalar replacementの最適化にLazy Code Motionのアルゴリズムを用いる場合、このDown-Safetyという集合情報を利用することにより、「メソッドの先頭にグローバルデータのアクセスを移動できる」と判断できるアクセスの集合を求めることができる。そして、この情報をグローバルデータに対する特殊化に用いる。

【００６５】

次に、影響度解析部１２が、汎用コンパイル処理部１１によるコンパイル結果（特殊化されていないオブジェクトプログラム）に対して影響度解析処理を行う（図２、ステップ２０２参照）。

影響度解析処理としては、まず、「メソッドの先頭にグローバルデータのアクセスを移動できる集合」が「DownSafety[メソッドの先頭]」に入っていると仮定し、この結果が格納されたローカル変数を定義する。

この後、定義されたローカル変数に対して、予め影響度解析の対象として設定された特殊化の各タイプ（図３参照）に対して、特殊化を行った場合における効果を見積もり、特殊化対象データを決定して影響度情報を作成する。これらの動作の内容は、（１）パラメータに対する特殊化の場合と同様である。

【００６６】

図１７は、上述したグローバルデータに対する特殊化における影響度解析処理

のアルゴリズムを説明する図である。

図１７のアルゴリズムにおける３、４行目の

“for (each p ∈ DownSafety[メソッドの先頭])”

“lvar = pの結果が格納されたローカル変数;”

においてグローバルデータのローカル変数化を実現している。また、８行目の

“Estimate(impact, lvar, lvar, φ);”

の処理の内容は、図５に示したパラメータの特殊化における特殊化の効果を見積もる処理と同様である。

【００６７】

次に、特殊化データ選択部１３が、影響度解析部１２による影響度解析処理の結果に基づいて、統計処理を行い、特殊化を行うグローバルデータ及びその値（実行データ）を決定する（図２、ステップ２０３、２０４参照）。これらの処理は、上述したパラメータの特殊化における特殊化データ選択部１３の処理と同様である。

【００６８】

最後に、特殊化コンパイル処理部１４が、特殊化データ選択部１３により選択された特殊化実行データに対して、実際に特殊化を実行する。特殊化は、上述したように、パラメータの種類や特殊化実行データの値に応じて個別的に実行される。

【００６９】

次に、パラメータ及びグローバルデータの特殊化の適用例を説明する。

図１８は、本適用例に用いるサンプルプログラム（以下、第３のサンプルプログラムと記す）を示す図である。

コンパイラ１０に図１８の第３のサンプルプログラムが入力されると、まず、汎用コンパイル処理部１１により、当該第３のサンプルプログラムに対して通常

のコンパイルが行われる。

図１９は、図１８の第３のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【００７０】

次に、影響度解析部１２により、汎用コンパイル処理部１１によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。ここでは、上述したパラメータに対する影響度解析処理と、グローバルデータに対する影響度解析処理とが区別なく実行される。

図２０は、図１９に示したサンプルプログラム３のコンパイル結果に対する影響度解析処理の結果を示す図である。

図２０を参照すると、パラメータfromIndexが何らかの定数値を取る場合に、影響度値１．７５が得られる。また、パラメータthis．offsetが何らかの定数値を取る場合に、影響度値５．７５が得られる。また、パラメータthis．countが何らかの定数値を取る場合に、影響度値５．７５が得られる。したがって、図２０に提示された５つのパラメータのうち、fromIndex、this．offset及びthis．countの３つについて、定数値の出現頻度の統計を取ればよいことがわかる。そして、この３つのパラメータに関する情報が特殊化対象データとして影響度情報に記述される。

なお、図１９に示したように、汎用コンパイル処理部１１のコンパイルによってサンプルプログラム３におけるグローバルデータであるthis．offset やthis．countがローカル変数に変換されているため、図２０においては、これらのパラメータが他のパラメータchなどと区別なく扱われていることがわかる。

また、図１９のプログラムにおいて太字で記し下線を付した部分が、パラメータfromIndex、this．offset及びthis．countの位置である。

【００７１】

次に、特殊化データ選択部１３により、パラメータfromIndex、this．offset及びthis．countに関して、取り得る値ごとに出現頻度の統計が取られる。そして、得られた統計情報に基づいて特殊化実行データが決定される。具体的な統計情報の提示は省略するが、ここでは、fromIndexは定数値０、this．offsetは定

数値０、this．countは定数値４を取る場合がそれぞれ最も出現率が高く、この場合の出現率と影響度値との乗算値はガードコードの挿入による遅延分の１よりも大きいものとする。したがって、パラメータfromIndex、this．offset及びthis．countは特殊化実行データとされる。

【００７２】

以上の処理の後、特殊化コンパイル処理部１４により、図１９に示した通常のコンパイル後のサンプルプログラム３におけるfromIndexが値０に、this．offsetが値０に、this．countが値４にそれぞれ特殊化されてコンパイルされる。

図２１は、図１９のプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【００７３】

【発明の効果】

以上説明したように、本発明によれば、影響度解析処理及び統計処理を経ることにより、コンパイルに要する時間を短縮し、かつコンパイル時のメモリ消費量を抑えることができる。

また、本発明によれば、メソッドコールを行う場所で呼び出しメソッドが特定できない場合であっても特殊化を実行可能とすることができる。

【図面の簡単な説明】

【図１】　本実施の形態におけるコンパイラの全体構成を説明する図である。

【図２】　本実施の形態によるコンパイラの全体的な処理の流れを説明するフローチャートある。

【図３】　本実施の形態における特殊化のタイプと特殊化を行う命令との対応関係を示す図表である。

【図４】　本実施の形態のパラメータの特殊化における影響度解析処理のアルゴリズムを説明する図である。

【図５】　本実施の形態のパラメータの特殊化における特殊化の効果を見積もる処理のアルゴリズムを説明する図である。

【図６】　本実施の形態のパラメータの特殊化における特殊化データ選択部

による統計処理のアルゴリズムを説明する図である。

【図７】　本実施の形態のパラメータの特殊化における特殊化実行データを選択する処理のアルゴリズムを説明する図である。

【図８】　本実施の形態のパラメータの特殊化における一適用例に用いるサンプルプログラムを示す図である。

【図９】　図８に示したサンプルプログラムに対する影響度解析処理の結果を示す図である。

【図１０】　図８に示したサンプルプログラムに対する統計情報の例を示す図である。

【図１１】　図８のサンプルプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

【図１２】　本実施の形態のパラメータの特殊化における他の適用例に用いるサンプルプログラムを示す図である。

【図１３】　図１２のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【図１４】　図１２に示したサンプルプログラムに対する影響度解析処理の結果を示す図である。

【図１５】　図１２のサンプルプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

【図１６】　図１２のサンプルプログラムに対してさらに他の条件で特殊化が行われた状態のソースプログラムを示す図である。

【図１７】　本実施の形態のグローバルデータに対する特殊化における影響度解析処理のアルゴリズムを説明する図である。

【図１８】　本実施の形態のグローバルデータに対する特殊化における一適用例に用いるサンプルプログラムを示す図である。

【図１９】　図１８のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【図２０】　図１９に示したサンプルプログラムのコンパイル結果に対する影響度解析処理の結果を示す図である。

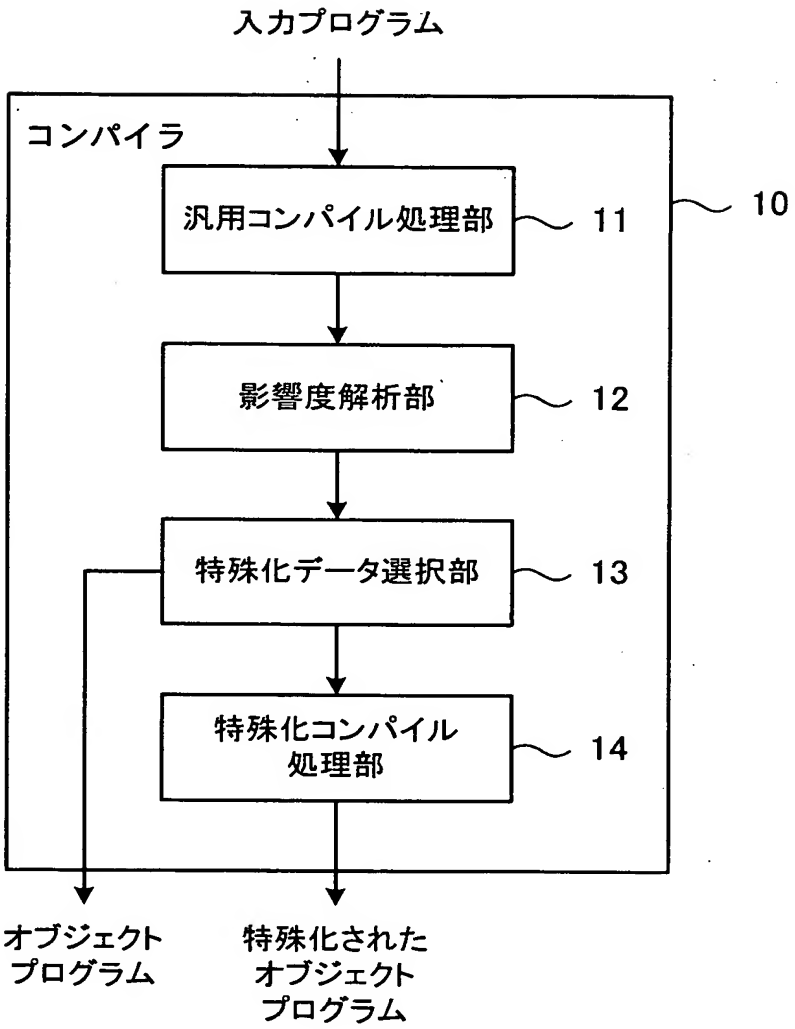
【図２１】　図１９のプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

【符号の説明】

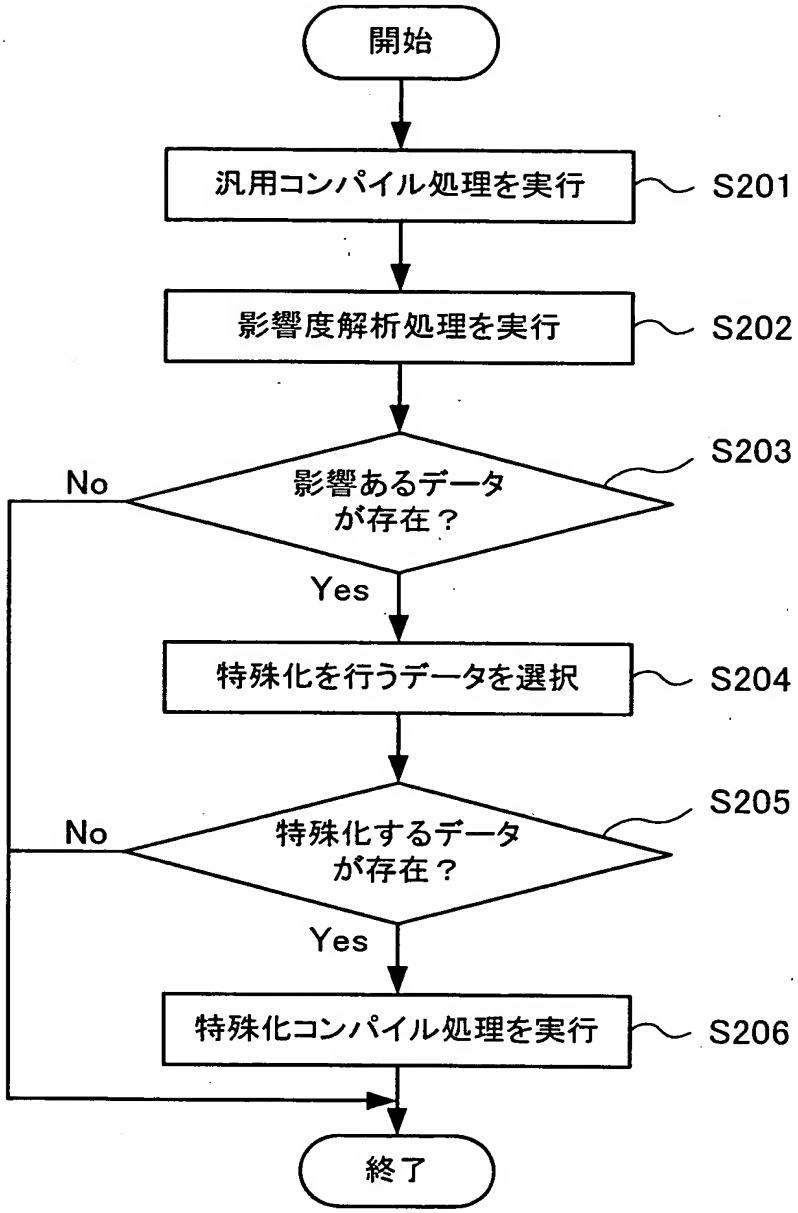
１０…コンパイラ、１１…汎用コンパイル処理部、１２…影響度解析部、１３…特殊化データ選択部、１４…特殊化コンパイル処理部

【書類名】　　図面

【図１】


入力プログラム
コンパイラ
汎用コンパイル処理部
11
10
影響度解析部
12
特殊化データ選択部
13
特殊化コンパイル
処理部
14
オブジェクト
プログラム
特殊化された
オブジェクト
プログラム

【図２】


開始
汎用コンパイル処理を実行
S201
影響度解析処理を実行
S202
影響あるデータ
が存在？
S203
No
Yes
特殊化を行うデータを選択
S204
特殊化するデータ
が存在？
S205
No
Yes
特殊化コンパイル処理を実行
S206
終了

【図３】

タイプ	命令
定数	四則演算 オブジェクト以外の変数に対する比較命令（switch命令も含む）
Nullではない	変数がNullかどうかを明示的にチェックする命令全般
クラスの特定	INSTANCEOF CHECKCAST 仮想的なメソッド呼び出し命令全般
配列以外のクラスである	オブジェクトが配列かどうかをチェックする命令全般．具体的には INSTANCEOF CHECKCAST INVOKEVIRTUALOBJECT_QUICK

出証特２００１－３０５８８４３

特２００１－０５５９９６

【図４】

```
Effective = φ;
for (each p ∈ 全てのパラメータ) {
    for (each type ∈ ALLTYPE) {
        Impact[p][type] = 0;
    }

    Estimate(impact, p, p, φ);

    for (each type ∈ ALLTYPE) {
        if (Impact[p][type] > しきい値A) {/* しきい値が１の時ガードコードと同等 */
            Effective ∪= { p, type };
        }
    }
}
```

【図５】

```
Estimate(impact, v, p, type)
{
   for (each l ∈ vを使っている命令) {     /* DU-chainを使う */
       if (l上のvの定義はただ一つ) {       /* UD-chainを使う */
           tmp_type = ALLTYPEの中のどの特殊化を行えば，v によって命令lが最適化されるか; - (1)
           if (tmp_type != φ) { /* v によって l が最適化されるならば */
               if (type == φ) {
                   type = tmp_type;
               }
               Impact[p][type] += typeに対するガードコードのコストを１としたときの，命令lが最
               適化されたときの影響度;  /* 例えば，2ならばガードコードの2倍・0.5ならばガード
               コードの半分の効果がある */
               if (特殊化によりlが定数への代入命令になる) {
                   Estimate(impact, lの結果が入るローカル変数, p, type);
               }
           }
       }
   }
}
```


出証特２００１－３０５８８４３

特２００１－０５５９９６

【図６】

```
if (Effective == φ) パラメータの特殊化の処理を中断
for (each { p, type } ∈ Effective) {
    パラメータ p について特殊化の方法 type の条件で統計を取る.
}
```

【図７】

```
Specialize = φ;
for (each { p, type } ∈ Effective) {
    odds = { p, type }に関する統計の中で，もっとも優秀だった確率;
    val = oddsに対応する{ p, type }の値;
    if (Impact[p][type] * odds > しきい値B) { /* しきい値が１の時ガードコードと同等 */
        Specialize U= { p, type, val };
    }
}
if (Specialize == φ) パラメータの特殊化の処理を中断
```

【図８】

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcBegin < 0) {
        throw new StringIndexOutOfBoundsException(srcBegin);
    }
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (srcBegin > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);
    }
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,
                     srcEnd - srcBegin);
}
```

【図９】

（Ａ）

影響度解析が終わったときのImpact[p][type]

p	type	Impact[p][type]
srcBegin	定数	2.25
	それ以外	0
srcEnd	定数	1.25
	それ以外	0
dst	全て	0
dstBegin	全て	0

（Ｂ）

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcBegin < 0) {
        throw new StringIndexOutOfBoundsException(srcBegin);
    }
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (srcBegin > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);
    }
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,
                     srcEnd - srcBegin);
}
```

【図１０】

SPECjvm98のJackベンチマークを使った場合の統計

srcBegin	値０	確率 100%
srcEnd	値１	確率 68%
	値０	確率 15%
	値２	確率 4%
	値３	確率 3%
	値４	確率 3%
	（以下略）	

【図１１】

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (0 > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    System.arraycopy(this.value, this.offset, dst, dstBegin, srcEnd);
}
```

【図１２】

```
boolean dispatchEvent(AWTEvent e) {
    boolean ret = false;
    if ((e instanceof MouseEvent) &&
        ((eventMask & MOUSE_MASK) != 0)) {
        MouseEvent me = (MouseEvent) e;
        ret = processMouseEvent(me);
    } else if (e instanceof FocusEvent) {
        FocusEvent fe = (FocusEvent) e;
        ret = processFocusEvent(fe);
    } else if (e instanceof KeyEvent) {
        KeyEvent ke = (KeyEvent) e;
        ret = processKeyEvent(ke);
    }
    return ret;
}
```

【図１３】

```
boolean dispatchEvent(AWTEvent e) {
    boolean ret = false;
    if ((e instanceof MouseEvent) &&
        ((eventMask & MOUSE_MASK) != 0)) {
        MouseEvent me = e;      /* checkcastが除去された */
        ret = processMouseEvent(me);
    } else if (e instanceof FocusEvent) {
        FocusEvent fe = e;      /* checkcastが除去された */
        ret = processFocusEvent(fe);
    } else if (e instanceof KeyEvent) {
        KeyEvent ke = e;  /* checkcastが除去された */
        ret = processKeyEvent(ke);
    }
    return ret;
}
```

【図１４】

影響度解析が終わったときのImpact[p][type]

p	type	Impact[p][type]
e	クラスの特定	3.94
	それ以外	0

【図１５】

```
boolean dispatchEvent(AWTEvent e)  {
    boolean ret = false;
    if (e != null) {
       KeyEvent ke = e;
       ret = processKeyEvent(ke);
    }
    return ret;
}
```

【図１６】

```
boolean dispatchEvent(AWTEvent e)  {
    boolean ret;
    KeyEvent ke = e;
    ret = processKeyEvent(ke);
    return ret;
}
```

【図１７】

```
Effective = φ;
ALLTYPE = 全ての特殊化するタイプ
for (each p ∈ DownSafety[メソッドの先頭]) {
    lvar = pの結果が格納されたローカル変数;
    for (each type ∈ ALLTYPE) {
        Impact[p][type] = 0;
    }

    Estimate(impact, lvar, φ);

    for (each type ∈ ALLTYPE) {
        if (Impact[p][type] > しきい値A) { /* しきい値が１の時ガードコードと同等 */
            Effective ∪= { p, type };
        }
    }
}
```

【図１８】

```
public int indexOf(int ch, int fromIndex) {
    int max = this.offset + this.count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= this.count) {
        return -1;
    }
    for (int i = this.offset + fromIndex ; i < max ; i++) {
        if (v[i] == ch) {
            return i - this.offset;
        }
    }
    return -1;
}
```

【図１９】

```
public int indexOf(int ch, int fromIndex) {
    int offset = this.offset;
    int count = this.count;
    int max = offset + count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= count) {
        return -1;
    }
    for (int i = offset + fromIndex ; i < max ; i++) {  -- (1)
        if (v[i] == ch) {
            return i - offset;
        }
    }
    return -1;
}
```

【図２０】

影響度解析が終わったときのImpact[p][type]

p	type	Impact[p][type]
ch	全て	0
fromIndex	定数	1.75
	それ以外	0
this.offset	定数	5.75
	それ以外	0
this.count	定数	5.75
	それ以外	0
this.value	全て	0

【図２１】

```
public int indexOf(int ch, int
fromIndex) {
    char v[] = this.value;

    if (v[0] == ch) return 0;
    if (v[1] == ch) return 1;
    if (v[2] == ch) return 2;
    if (v[3] == ch) return 3;
    return -1;
}
```

【書類名】　　要約書

【要約】

【課題】　特殊化による最適化を含むコンパイルにおいて、コンパイルに要する時間を短縮し、コンパイル時のメモリ消費量を抑える。

【解決手段】　コンパイラ１０において、コンパイルの対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部１２と、このプログラムを実行した場合に、この影響度解析部１２にて解析された命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいてこの命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部１３と、この影響度解析部１２及び特殊化データ選択部１３による処理結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化コンパイル処理部１４とを備える。

【選択図】　　図１

認定・付加情報

特許出願の番号	特願２００１－０５５９９６
受付番号	５０１００２８７１６２
書類名	特許願
担当官	風戸　勝利　　９０８３
作成日	平成１３年　４月２０日

＜認定情報・付加情報＞

【特許出願人】
- 【識別番号】 390009531
- 【住所又は居所】 アメリカ合衆国１０５０４、ニューヨーク州　アーモンク　（番地なし）
- 【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】
- 【識別番号】 100086243
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　大和事業所内
- 【氏名又は名称】 坂口　博

【代理人】
- 【識別番号】 100091568
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　大和事業所内
- 【氏名又は名称】 市位　嘉宏

【代理人】
- 【識別番号】 100106699
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番１４　日本アイ・ビー・エム株式会社大和事業所内
- 【氏名又は名称】 渡部　弘道

【復代理人】 申請人
- 【識別番号】 100104880
- 【住所又は居所】 東京都港区赤坂５－４－１１　山口建設第２ビル６Ｆ　セリオ国際特許事務所
- 【氏名又は名称】 古部　次郎

【選任した復代理人】
- 【識別番号】 100100077

次頁有

認定・付加情報（続き）

【住所又は居所】　東京都港区赤坂５－４－１１　山口建設第２ビル
６Ｆ　セリオ国際特許事務所
【氏名又は名称】　大場　充

次頁無

出　願　人　履　歴　情　報

識別番号　　〔３９０００９５３１〕

１．変更年月日　　２０００年　５月１６日
　［変更理由］　名称変更
　　住　所　　アメリカ合衆国１０５０４、ニューヨーク州　アーモンク　（番地なし）
　　氏　名　　インターナショナル・ビジネス・マシーンズ・コーポレーション